Programming Assignment 6
Vacation Account Balance

Time due: 9:00 PM, March 16th (Extended to March 19th)

Your task

Your assignment is to produce a two classes that work together to simulate BloodDonation and a VacationAccount. To help you, the design of these two classes will be discussed here. Please follow the steps outlined below and don't jump ahead until you have finished the earlier step.

First, you will create the class BloodDonation. Each BloodDonation object has an employee ID number, the employee's age, and the employee's weight. All three of these parameters are provided to the BloodDonation constructor. In addition to a constructor, each data member has a public accessor and mutator operation. The constructor and the mutator operations should enforce the following data validation rules:
- a valid employee ID number must be a six digits integer number. Each digit must be in the range of 0 to 9. A value of -1 will be stored as the employee ID number when the number provided is an invalid number.
- a valid employee's age must be an integer value ranging from 21 to 65. The value of -1 will be stored when the age provided is outside of the valid range.
- a valid employee's weight must be a double value ranging from 101.00 to 280.00. The value of -1 will be stored when the weight provided is outside of the valid range.

Please review the following:

- mID: integer
- mAge : integer
- mWeight : double

+ BloodDonation(EmployeeID : integer, EmployeeAge: integer, EmployeeWeight: double)

+ getID() : integer
+ setID() : integer

+ getAge() : integer
+ setAge() : integer

+ getWeigt() : double
+ setWeight() double

Next, create the VacationAccount class. Each VacationAccount object has an employee ID number associated with the account and its vacation balance. Solely the employee ID number parameter is provided to the VacationAccount constructor. In the beginning of time, the balance should start at zero. In addition to a constructor, each data member has a public accessor operation. The vacation balance gets increased by 4 hours every time the employee donates blood. The increase is done via calls to .addVacationToAccount(...).

.addVacatioToAccount (...) should return true when the:

- employee ID number matches the VacationAccount employee ID number, and the employee ID number is not -1.
- Employee's age and weight are not -1.

.addVacatioToAccount (...) returns false otherwise. No increase of vacation balance is allowed if false.

Please review the following:

- mID: integer
- mBalance: double

+ VacationAccount(EmployeeID : integer)

+ getBalance() : double
+ getID() : integer

+ addVacationToAccount(donation : BloodDonation) : bool

For this project, you will create both a .h and .cpp files. Write some sample driver code in your main( ) and create assertions to verify that your accessor methods are all working properly. Some sample code is shown below to further document how this class should work.

You are free to create additional public and private methods and data members as you see fit. However, the test cases will only be driving the public methods of the two classes illustrated above.

The source files you turn in will be these classes and a main routine. You can have the main routine do whatever you want because we will rename it to something harmless, never call it, and append our main routine to your file. Our main routine will thoroughly test your functions. You'll probably want your main routine to do the same. If you wish, you may write additional class operations in addition to those required here. We will not directly call any such additional operations directly.

The program you turn in must build successfully, and during execution, no method may read anything from cin.

Please read the posted FAQ for further assistance.

Programming Guidelines

Your program must *not* use any function templates from the algorithms portion of the Standard C++ library or use STL <list> or <vector>. If you don't know what the previous sentence is talking about, you have nothing to worry about. Additionally, your code must *not* use any global variables which are variables declared outside the scope of your individual functions.

Your program must build successfully under both Visual C++ and either clang++ or g++.

The correctness of your program must not depend on undefined program behavior.

What you will turn in for this assignment is a zip file containing the following 6 files and nothing more:

1. The text files named **BloodDonation.h** and **BloodDonation.cpp** that implement the BloodDonation class illustrated above, the text files named **VacationAccount.h** and **VacationAccount.cpp** that implement the VacationAccount class illustrated above, and the text file named **main.cpp** which will hold your main program. Your source code should have helpful comments that explain any non-obvious code.
2. A file named **report.doc** or **report.docx** (in Microsoft Word format), or **report.txt** (an ordinary text file) that contains in addition **your name** and **your UCLA Id Number**:
   - A brief description of notable obstacles you overcame
   - A list of the test data that could be used to thoroughly test your functions, along with the reason for each test. You must note which test cases your program does not handle correctly. (This could happen if you didn't have time to write a complete solution, or if you ran out of time while still debugging a supposedly complete solution.) Notice that

most of this portion of your report can be written just after you read the requirements in this specification, before you even start designing your program.

Your report this time doesn't have to contain any design documentation.

As with Project 3 and 4, a nice way to test your functions is to use the assert facility from the standard library. As an example, here's a very incomplete set of tests for Project 5. Again, please build your solution incrementally. So I wouldn't run all these tests from the start because many of them will fail until you have all your code working. But I hope this gives you some ideas....

```
#include <iostream>
#include <string>
#include <cassert>

#include "BloodDonation.h"
#include "VacationAccount.h"

using namespace std;

int main()
{
// sample test code
BloodDonation doner1( 752401, 45, 99.56);

BloodDonation doner2( 889543, 55, 109.50);

BloodDonation doner3( 89643244, 65, 187.55);

BloodDonation doner4( 855565, 17, 127.00);

VacationAccount account(889543);

assert( std::to_string(doner1.getID( )) == “752401”);

assert(std::to_string(doner2.getAge( )) == “55” );

assert(std::to_string(doner3.getWeight( )) == “187.55” );

// account balance starts at zero...

assert( std::to_string(account.getBalance( ) ) == "0.000000" );

assert( std::to_string(account.getID( )) ==  “889543”);
```

```
    // Vacation adds to vacation balance

    assert( account.addVacationToAccount( doner2 ) == true );

    assert( account.addVacationToAccount( doner1 ) == false );

    assert( std::to_string( account.getBalance( ) ) == "4.000000" );
```

G31 Build Commands

```
g31 -c BloodDonation.cpp
g31 -c VacationAccount.cpp
g31 -c main.cpp
g31 -o runnable main.o BloodDonation.o VacationAccount.o
./runnable
```